Node Embeddings for Graph Merging

Node Embeddings for Graph Merging: Case of Knowledge Graph Construction

Ida Szubert **Mark Steedman**
School of Informatics
University of Edinburgh
Edinburgh, Scotland, UK
k.i.szubert@sms.ed.ac.uk, steedman@inf.ed.ac.uk

Abstract

Combining two graphs requires merging the nodes which are counterparts of each other. In this process errors occur, resulting in incorrect merging or incorrect failure to merge. We find a high prevalence of such errors when using AskNET, an algorithm for building Knowledge Graphs from text corpora. AskNET node matching method uses string similarity, which we propose to replace with vector embedding similarity. We explore graph-based and word-based embedding models and show an overall error reduction of from 56% to 23.6%, with a reduction of over a half in both types of incorrect node matching.

1 Introduction

The work we present here is an extension of the graph building algorithm of AskNET (Harrington and Clark, 2008). The overall task we consider is to automatically extract information from text and integrate it into one resource, a knowledge graph. In this paper we focus on one aspect of the AskNET graph building algorithm – the process of merging document-level graphs into a corpus level one. We propose improvements which reduce the rates of two common and non-trivial errors which arise when matching nodes between two graphs.

Merging two graphs, with an assumption that they potentially represent overlapping information, requires node matching. We need to decide which nodes should be merged, and this can result in two types of errors: merging two nodes even if they do not represent the same thing, or not merging nodes which do. As an example, Figure 1 shows a sample graph representing information from a news article and a new sentence together with its own graph. In merging the two graphs we could correctly decide to match and merge the *Williams* node to the *Rowan Williams* one. If we matched to *Justin Welby* we would merge the nodes incorrectly, resulting in conflation of information about two different people. If we did not find a match at all the result would be the presence of two nodes representing the same person. We call the first error spurious merging, and the second spurious addition.

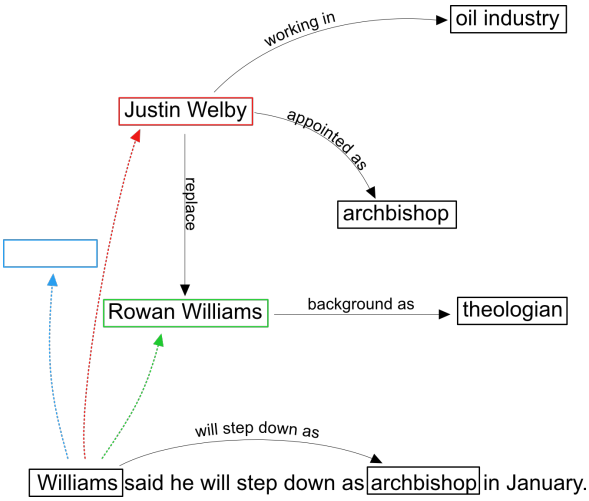

Figure 1: A graph representing a news article about Archbishop appointment and a sentence to be integrated into the graph. Green dotted arrow shows correct node matching, the red match leads to spurious merging, and the blue arrow and node represent no match found and a spurious addition.

Both of those problems occur in graphs built with the AskNET algorithm, which uses string similarity as a tool for node matching. We propose to modify the algorithm with using vector embeddings instead. We explore word-based and graph-based ways of obtaining graph node embeddings and show that both lead to reduction in merging and addition errors.

2 Methods

In this paper we explore a modification of the AskNET algorithm. We provide an overview of the KG-building approach of (Harrington and Clark, 2008), and a more detailed explanation of the step which we improve upon.

Proceedings of the Thirteenth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-13), pages 172–176
Hong Kong, November 4, 2019.

2.1 AskNET

The AskNET system uses a semantic parser (Boxer (Bos, 2005)) to obtain graph representation of sentential content in a large corpus and incrementally combines them, first into document level graphs (DG), and then into one semantic network, which we are calling a knowledge graph (KG)[1]. The central challenge in the process of graph merging is, as described before, node matching. In AskNET the matching problem is approached with an iterative message passing (aka activation spreading) procedure, the intuition behind which is to find, for a node n_{DG} from DG, a node n_{KG} in the KG such that the neighbourhoods of n_{DG} and n_{KG} in their respective graphs are similar.

Given a node n_{DG}, the algorithm selects a candidate set C_{KG} of KG nodes such that it likely contains the correct matching n_{KG}. Afterwards in an iterative procedure for each of the DG nodes the scores of each element of C_{KG} are updated until convergence. The pre-selection step is crucial for ensuring time and space efficiency of the algorithm.However it also means that if the quality of C_{KG} is a limiting factor for the correctness of the output. If the correct match is not present in C_{KG}, either spurious merging or addition will result. Moreover, errors will also arise when C_{KG} does contain the correct match, but it is easily confused with the rest of the set members.

As Harrington (2009) recognizes, spurious addition and merging are challenging errors to fix, and the AskNET algorithm does not attempt it. In our implementation of AskNET, only about 44% of nodes are correctly matched, with the rest divided between spurious addition (24%) and merging(32%). We approach this issue by improving the candidate node sets with the aim of minimizing the problems. The task we consider is thus: given a KG, a DG, and node n_{DG}, produce a set C_{KG} such that it (1) contains the true match and (2) minimizes merging and addition errors. In the approach of (Harrington and Clark, 2008) the selection is based on approximate string matching between the name associated with n_{DG} and the names associated with the KG nodes. This similarity measure is not reliable in contexts where one entity can be called by various names and titles, as is especially common in news text in which repetitions are avoided. In our evaluation, only about 55% of candidate sets generated with string similarity actually contain a correct match. Our contribution is investigating the use of vector embeddings for candidate node selection. We show that both word-based and graph-based embedding models provide a better notion of similarity, and that combining them brings optimal benefits.

2.2 Dataset

For our experiments we use a subsection of the NewsSpike corpus (Zhang and Weld, 2013). For purposes of efficiency and ease of evaluation we decided to build a KG of around 15k nodes (size in line with the FB15k dataset (Bordes et al., 2013) popular in graph embedding literature), which we achieved by selecting 127,221 documents from the NewsSpike corpus. We preprocessed the NewsSpike corpus using the DBPedia entity linker (Nguyen et al., 2014) which enabled us to identify the most frequently occurring named entities. We chose top n named entities such that when a KG is created of all the documents mentioning those n entities, the graph has approximately 15k nodes.[2] Documents in our dataset are relatively short, averaging 338 words. The average number of named entities per document is 10.3. We held out 10 randomly selected documents each for development and test sets, and the rest forms the training set.

2.3 Base KG

The KG used in our experiments is built out of the training set documents using the original AskNET algorithm. To obtain graphs representing individual sentences we use the semantic parser of (Reddy et al., 2014) and extract binary relations from its output. We only use relations which involve at least one named entity. The nodes in the graph are labeled with a set containing all strings from the original documents which have been linked to that node (e.g. a set containing *Rowan Williams, Williams, Rev. Williams, Archbishop of Canterbury*). The edges are labeled with relation names, where the relation set is open and can include any two place predicate used in the source documents.

[1] Semantic parses used in AskNET follow a version of Discourse Representation Theory as implemented by the Boxer parser; in our replication we represent the information contained by a sentence with a set of entity-relation triples, which express all the binary relations involving at least one named entity

[2] The graph also includes nodes representing non-named entities.

2.4 Graph-based embeddings

The first way of generating DG and KG node embeddings is using a graph embedding method. We decided to use the *GraphSAGE* model of (Hamilton et al., 2017) which generates node representations by taking into account the features of each node and the structure of it's neighborhood. This model, as compared to numerous recent graph embedding models, is particularly well suited to our use case. It can be trained with an unsupervised objective, once trained can produce embeddings for unseen nodes and even nodes in unseen graphs, and leverages node features, e.g. text attributes. We train GraphSAGE, in an unsupervised setting[3] and mean aggregators, on the base KG. The initial node features are node degree and mean of one-hot encoding of the node labels. The model learns a set of aggregator functions, which not only generate final embeddings for the KG nodes, but also for DG nodes in the development and test sets.

2.5 Word-based embeddings

Another approach to obtaining node embeddings is through word embeddings. We make use of of the ELMo model for deep contextualized word representation (Peters et al., 2018). We represent a node in a KG as an average of the word embeddings of every entity that has been resolved to that node during graph building. In other words, given a document we generate ELMo embeddings for every mention of every entity[4], and in the DG a node representing an entity is assigned an embedding which is an average of all the mentions of that entity. When nodes are merged during integration of the DG into the KG, the embedding of the KG node is updated so that it is always an average of the embeddings of all document-level nodes resolved to that KG node. For the purposes of our experiments we perform this embedding aggregation when building the base KG so that embeddings for all of its nodes are available. However, we do not use those embeddings to aid in the building process.

We use the original large pre-trained ELMo model (**ELMo large**), which has been trained on the same genre as our corpus. For the purposes of meaningful comparison with the graph-based embeddings, which cannot be pre-trained, we also train an ELMo model on our corpus only (**ELMo small** and exeriment with the resulting embeddings.

2.6 Hybrid embeddings

We expect that the two methods might provide complementary benefits: GraphSAGE make use of the structure of the KG, and ELMo accumulate information from the original texts, including information which is not present in the KG. We propose to combine them by using the **GraphSAGE** model with initial node features being node degree and the **ELMo large** embedding of the node.

3 Experiments

We propose to find the set C_{KG} by evaluating embedding similarity, rather than string similarity, between embeddings of the n_{DG} and all KG nodes. We treat the string-similarity method as a baseline. We expect relying on embedding similarity to result in better candidate sets and in reduced number of merging and addition errors.

Regardless of the embedding method, candidate selection requires us to pick k closest neighbours for a given node from all of the nodes of the KG. To do that we use random projection-based approximate nearest neighbour search algorithm implemented in the Annoy library[5]. For candidate selection in the original string-based method we use the SimString library which performs approximate string matching according to the method proposed in (Okazaki and Tsujii, 2010), and we define the similarity between n_d and n_{KG} as the edit distance divided by the length of the shorter of the names. We use the development set to set k to 7 and string similarity threshold to 0.3[6].

For each test document, we build a DG using the original AskNET method. Each node in that graph is associated with a set of names and ELMo large and ELMo small embeddings. Using the GraphSAGE and hybrid models trained on the base KG we assign each node in DG a GraphSAGE and hybrid embeddings. Then, for each DG node we find five C_{KG} sets, one for each method, and run the rest of the AskNEt algorithm for each set to

[3] with the objective of maximizing the similarity of close by nodes and minimizing that of distant nodes, where the closeness is determined by co-occurrence on fixed-length random walks

[4] Because of pre-processing the documents in our corpus with an entity linker we know which word sequences constitute entity names and we treat them as single lexical item.

[5] https://github.com/spotify/annoy

[6] We also use the development data to set various AskNET hyperparameters not discussed in this work

	good C_{KG}	correct	errors merging	errors addition
baseline	54.5	44.2	24.2	31.6
GraphSAGE	67.3	61.8	18.2	20.0
ELMo small	63.6	55.1	19.4	25.5
ELMo large	71.5	64.2	13.9	21.9
hybrid	**82.4**	**76.4**	**10.9**	**12.7**

Table 1: Node matching results: percentage of C_{KG} containing a correct match (higher is better); percentage of test nodes correctly resolved (higher is better), spuriously merged with some KG node and spuriously added to KG as a new node (lower is better)

recover the one (or none) final match for each method.

3.1 Evaluation

Evaluation of information graph building methods is inherently challenging in the absence of gold-standard KGs. In our experimental setting the base KG is automatically constructed and as such is noisy. We then ask how well are different methods capable of matching DG nodes to the nodes in the noisy KG. There is no ground truth, and so our evaluation relies on a human annotator inspecting the neighbourhoods of the candidate KG nodes and making a judgment about how likely is it that they are a match to the DG node. When the annotator finds no likely matches in the set, they search the KG (by keywords, names, etc.) to ascertain whether likely matches are present in the KG at all. The annotator has access to the test set documents, document-level graphs, and can inspect the KG.

For each node in a DG in the test set, we manually evaluate the candidate sets produced by the five methods (baseline, GraphSAGE, ELMo large, ELMo small, hybrid) and the one (or none) node returned by the AskNET algorithm as the final match. There are 165 nodes in the 10 test set documents.

We evaluate the candidate sets by manually assessing whether any of the candidates can be reasonably considered to be a match, and if so the set is deemed to be correct. For the final match decision we perform a 3-way classification: correct KG node selected or correct in choosing no KG node; incorrect KG node selected and leading to spurious merging; or incorrectly selecting no KG node and leading to spurious addition.

4 Results

Overall, both graph- and word-based embedding similarity outperform string similarity as a candidate node selection tool, and the most benefit is to be had by using a hybrid model. Both the percentage of good candidate sets and the percentage of correct matches are significantly increased. Moreover, the difference between the two measures is smaller for our proposed methods than for baseline, indicating that our candidate sets are less confounding for the AskNET algorithm.

As can be seen in table 1 word- and graph-based embedding differ in what kind of confounding candidates they introduce to the sets alongside the correct match. This is reflected in the different rates of spurious merging and addition they result in. GraphSAGE reduces addition more then merging. Given a node n_{DG}, using GraphSAGE we select KG nodes whose neighbourhood is similar, in terms of features and structure, to that of n_{DG}. This makes it more likely that the correct match is in C_{KG}, thus reducing spurious addition. However, the subsequent steps of the AskNET algorithm also rely on estimating the similarity of the neighborhoods which means that the candidates selected using this method are easy to confuse.

An opposite tendency can be observed for ELMo embeddings, where C_{KG} includes entities mentioned in similar textual context as n_{DG}. If the correct match is present in the set, it is easier for the AskNET algorithm to identify it using the neighbourhood information, because the candidates are likely to have diverse neighbourhoods. Therefore merging mistakes are significantly reduced.

5 Conclusions

We show that modern context-aware word embeddings and graph-based embeddings can both be used, separately or in conjunction, to improve KG building from text. Our experiments show that we can reduce both of the two difficult problems with resolving entities to KG nodes: spurious merging of separate entities into one node and spurious addition of nodes for entities which are already represented in the KG. We explore how different embedding methods are better suited to solving one of the problems than the other, and how combining them provides synergistic effects. While we explore the proposed methods from the point of view of KG creation, the same methods could be used for other KG-based tasks, e.g. question answering.

References

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795.

Johan Bos. 2005. Towards wide-coverage semantic interpretation. In *Proceedings of Sixth International Workshop on Computational Semantics IWCS*, volume 6, pages 42–53.

Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034.

Brian Harrington. 2009. *ASKNet: automatically creating semantic knowledge networks from natural language Text*. Ph.D. thesis, Oxford University.

Brian Harrington and Stephen Clark. 2008. Asknet: Creating and evaluating large scale integrated semantic networks. *International Journal of Semantic Computing*, 2(03):343–364.

Dat Ba Nguyen, Johannes Hoffart, Martin Theobald, and Gerhard Weikum. 2014. AIDA-light: High-Throughput Named-Entity Disambiguation. In *Workshop on Linked Data on the Web*, pages 1–10, Seoul, Korea.

Naoaki Okazaki and Jun'ichi Tsujii. 2010. Simple and efficient algorithm for approximate dictionary matching. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 851–859. Association for Computational Linguistics.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392.

Congle Zhang and Daniel S Weld. 2013. Harvesting parallel news streams to generate paraphrases of event relations. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1776–1786.